THE OFFICE OF THE STATE CHIEF INFORMATION OFFICER
ENTERPRISE TECHNOLOGY STRATEGIES

North Carolina Statewide Technical Architecture

Domain White Paper

System Integration Architecture Technology Overview

STATEWIDE TECHNICAL ARCHITECTURE

Domain White Paper:
System Integration Architecture Technology Overview

Initial Release Date:	August 1, 2003	Version:	1.0.0
Revision Approved Date:	Not Applicable		
Date of Last Review:	March 11, 2004	Version:	1.0.1
Date Retired:			
Architecture Interdependencies:			
Reviewer Notes: Reviewed and updated office title and copyright date. Added a hyperlink for the ETS email – March 11, 2004.			

Office of the State Chief Information Officer
Enterprise Technology Strategies
PO Box 17209
Raleigh, North Carolina 27699-7209
Telephone (919) 981-5510
ets@ncmail.net

Mission Statement

System Integration Architecture facilitates and simplifies communication within and between heterogeneous, distributed application systems.

Note: The System Integration Domain represents the incorporation of three previous chapters: Application Communication Middleware, Middleware, and Integration. The new consolidated domain will be revised in the near future to reflect the technical changes in the three chapters previously mentioned.

As the state moves to a distributed infrastructure, multiple applications must be able to exchange data across complex, heterogeneous environments; therefore, the state's technical infrastructure must be able to deliver pertinent information at the right place and time and in a useful format. In today's fast-paced and ever-changing environment, program-to-program is essential for departments to improve operations, offer better and new services, and reduce costs. Application communication middleware facilitates interchange of information in a distributed, multi-vendor, and heterogeneous systems environment while providing the same levels of security, reliability, and manageability traditionally associated with a monolithic, mainframe-based architecture where all products are supplied by a single vendor.

Middleware is an overused term in industry today; it seems that everything is some type of middleware. Middleware insulates application developers from having to understand the complexities of the computing environment. To programmers, middleware is a "black box," where understanding the details of what happens inside is not required. Just as high-level programming languages, such as COBOL, FORTRAN, and C insulate programmers from platform architecture, application communication middleware insulates programmers from the complexities of the communication architecture, such as network protocols.

There are two areas that require application communication middleware:

Intra-application. Handles communication within the tiers of an application system.

Inter-application. Handles communication between the application system and external services, such as common shared services and other application systems. (For more information on common shared services, refer to the Componentware Architecture chapter).

Both inter-application and intra-application communication middleware provide benefits:

Adaptability. The underlying components of the technical infrastructure (such as operating systems, databases, and hardware platforms) can be expanded or changed without having to modify the application systems that are supported by the infrastructure.

Reduced development time. The logical partitioning of an N-tiered system and the modularity of construction using common services offer efficiencies through the specialization of skills and the reusability of components. These practices will improve the quality of systems and reduce lead times for their implementation and modification.

Flexibility. The features and capabilities of applications can be modified (i.e., they are scaleable and expandable) without changing the foundation technical architecture.

Reduced costs. The opportunities for selecting products from different vendors are enhanced by the integration capabilities offered by middleware; therefore, greater competition will improve price offerings.

Intra-Application Communication Middleware

The Application Architecture explains how N-tier applications are divided into multiple tiers. (See Figure Below.) Intra-application communication middleware enables the tiers to communicate within an application system.

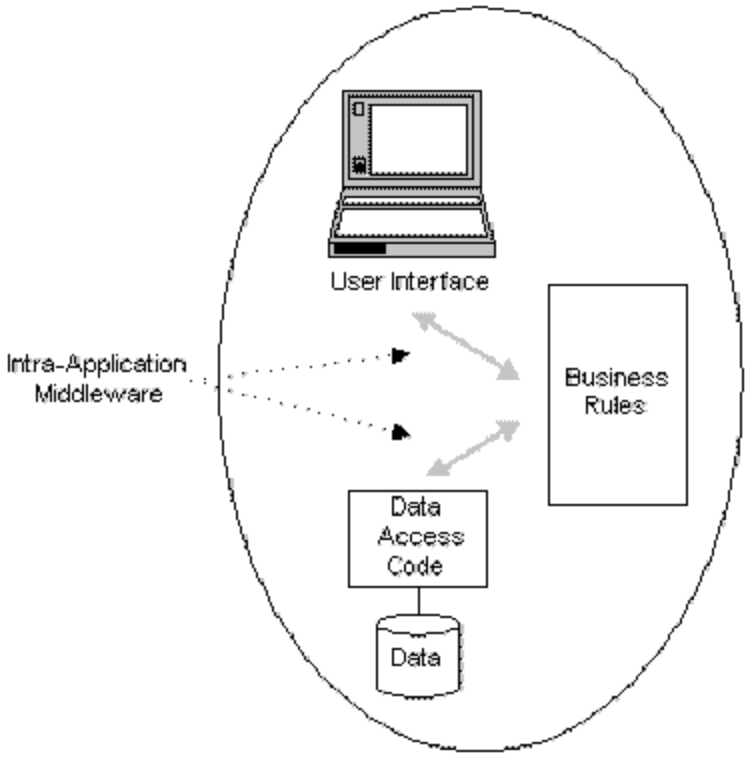

Figure 1. Communication between Application Tiers.

The logical tiers *within* a single application system often reside on different physical machines. Since the application tiers can reside on different platforms and physical machines, there needs to be communication and coordination among the logical tiers. Intra-application communication middleware simplifies the complexities associated with developing and deploying applications in a distributed, heterogeneous environment.

Inter-Application Communication Middleware

In addition to the communication requirements within an application system, inter-application communication middleware is required for communicating externally with services outside of an application system. Inter-application communication middleware facilitates the access to other application systems or common business services. Through inter-application communication middleware, applications can locate and interact with other applications or services on the network in a reliable and scaleable manner. (See Figure Below).

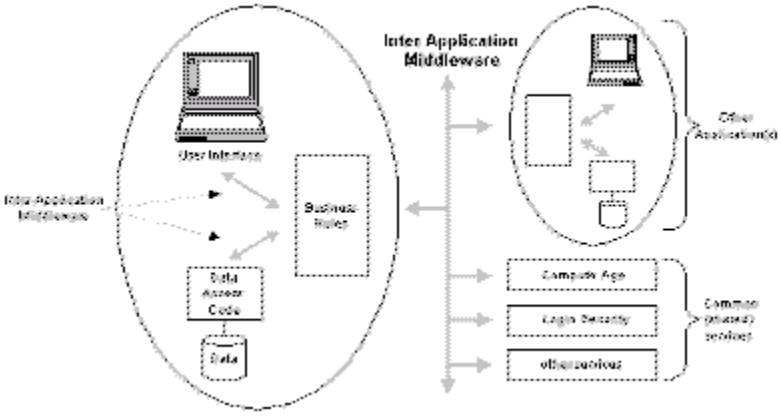

Figure 2. Inter-application Middleware

Benefits of Application Communication Middleware

Application communication middleware is the cornerstone of the state's application architecture. It must:

Maximize Flexibility in Managing Technical Infrastructure

Changes to the underlying state technical infrastructure must remain transparent to application programmers. This model permits changing the state's technical infrastructure (including middleware) with little or no modification to the supported applications. The model incorporating inter-application communication is called a "broker." A broker simplifies communication external to the application and insulates the underlying infrastructure. Source code does not have to change in response to changes in services or infrastructure. (See Figure Below).

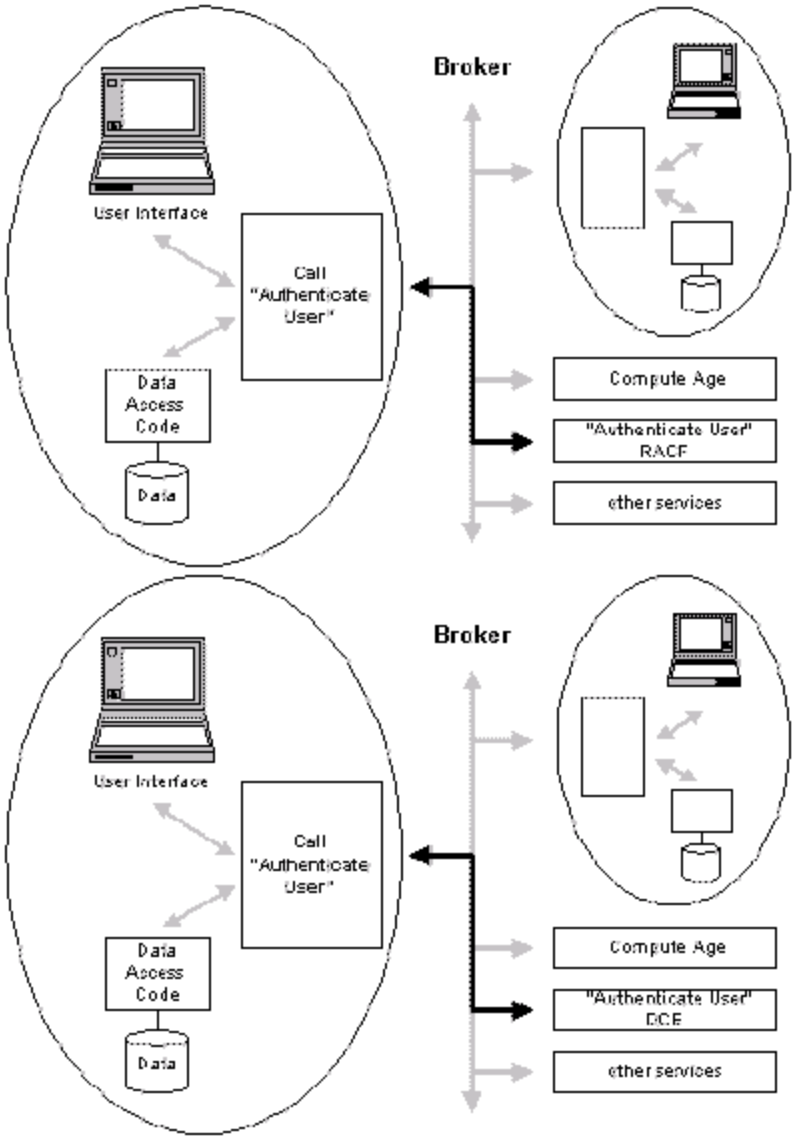

Figure 3. The Role of a Broker

As illustrated in Figure Above, user authentication is a sample core service needed by many applications. Today, application developers code an "Authenticate User" service in each application requesting user name and password information. Each "Authenticate User" service is application specific and is implemented to support the security system on the platform where the application runs. If a generic broker was used for the "Authenticate User" service, a common statewide authentication service can be provided. The application developer would only need a single security interface and would not need to know product-specific security information. In this way, the state can manage its technical infrastructure with

minimal, if any, impact on application developers. Expertise on specialized products, such as security, no longer is required of every application developer.

The use of a broker can facilitate timely implementation of legislation. For example, recent federal legislation required states to allow citizens to update their voter registration at the same time they apply for government services, such as a driver's license or unemployment benefits. If a broker had been in place at the time of the legislation, the change in legislation would have been far less complicated to implement. A broker would have prevented each agency's application developers from having to learn or build custom interfaces to the voter registration application system. A broker providing a common, consistent method of communication can streamline inter-application communication.

Minimize Complexity to Application Developers

Since much of the functionality in new applications will be provided by calling pre-written and pre-tested software services, such as "Compute Age," the services need to be easy to access. (For more information on reusable components, refer to the Componentware Architecture chapter). As software services, (e.g., "Compute Age"), become available and are documented in the component repository, they are available for application use through inter-application communication middleware. Application developers would call the "Compute Age" service when they needed that function, instead of developing and testing their own "Compute Age" function within their application.

Application systems also need the capability to communicate with other applications. Application systems within the same agency or even from different agencies may find that there is a need to interoperate to exchange information or services (e.g., the earlier example about federal legislation regarding voter registration. The legislation affected multiple agencies and application systems across the state).

Maximize Selection Availability of Development Tools

Unfortunately, there is no simple middleware solution that meets both intra- and inter-application communication needs; there is not a single communication paradigm. Since there are multiple methods of communication, a match must be made between the problem being solved and the type of middleware required. Selecting the right middleware product is not easy because there are no industry standards, and most middleware is proprietary. When proprietary solutions are used, interoperability is usually limited.

Often, middleware software is included with integrated development tools. Integrated CASE tool vendors and front end/back end tool vendors currently use

this approach. (For more information on development tools, refer to the Application Architecture chapter). The development tool vendor usually supplies its own proprietary application communication middleware. Development tools can also support third-party proprietary application communication middleware in addition to the proprietary middleware.

Applications developed in one development tool may not be able to easily communicate with applications developed in another tool. Standardizing on an application development tool and its associated communication middleware infrastructure will circumvent this problem.

If proprietary middleware interfaces are included in application systems, changes to the middleware infrastructure are impossible without also changing the applications. To minimize the risk of incompatible communications, third-party proprietary application communication middleware supported by a variety of tools should be selected. Third party middleware also permits some flexibility in selecting application development tools.

Selecting middleware within an application system poses much less risk for the state than selecting middleware for inter-application communication. Intra-application communication, used to communicate between the different tiers within an application system, only requires consistency within the same system. Therefore, there is no need for the statewide technical architecture to specify particular development tools or middleware for intra-application communication.

However, communicating between application systems is not the same as within an application system; a consistent solution for inter-application communication would greatly benefit the state. In addition to providing application developers with a single, consistent interface for inter-process communication, the underlying infrastructure can be isolated and more easily managed. One of the most important design principles for the technical architecture is that it must provide the flexibility for changes in technical structure to respond to changing technology in a cost-effective manner.

Integration Architecture

Mission Statement

Integration Architecture specifies how various automated applications operating on different platforms can effectively work together. Integration techniques should be used when new application systems need to access existing application systems, while maximizing the investment in existing systems and platforms. This chapter includes an introduction of integration, an explanation of data access techniques, application and 3270 terminal integration, and recommendations and standards for each component.

The state is in the process of optimizing applications and platforms so that application systems execute in a cost-effective manner and with performance and usage optimized. In doing so, the state is also faced with the challenge of integrating the new optimized client/server systems with established mainframe-based legacy systems and purchased software. Integration is the key to bridging the gap between heterogeneous operational application systems while still maximizing the investment in existing hardware and client platforms.

Integrating new client/server, adaptive, and distributed systems with existing systems while still optimizing performance, minimizing maintenance and utilizing existing platforms is a major technical challenge. When new client/server systems are developed, they need the ability to access the business processes and data from legacy and purchased systems developed under different technical architectures.

Integration is much easier when all of the applications are built on the same platform under the same standards. When applications do not use a common database management system (DBMS), data models or semantics, it becomes more difficult to integrate new systems with these applications. Many legacy application systems currently used by state agencies were developed independently using a wide variety of formats and designs because no enterprise standards were in place when the applications were developed. The state has also purchased vendor packages that need to be accessed by both legacy and new systems. Purchased software historically does not conform to state standards, especially relating to DBMS models and semantics. The integration strategy accommodates the coordination of legacy mainframe applications, purchased application packages, and newly developed *N*-tier client/server applications so they can work together.

An integration strategy can be implemented several ways, including:

Implicit integration. Exists from the end user perspective (e.g., the propagation of a change in address throughout multiple application systems). In the background, the systems are integrated through integration techniques that duplicate the processing, data transmission and data storage across all the application systems. The end users think that it is all one system even though it is not.

Explicit integration. When the application systems are tightly linked so that the processing, data transmission and data storage occurs only once (e.g., the address is stored in only one location for all systems). New application systems are usually designed this way. Explicit integration is further discussed in the Application Architecture chapter.

- Generally, enterprises integrate applications through a wide variety of solutions:
- Custom extract programs, transfer files and import programs.

- Screen-scraping programs.
- Custom-developed interface programs.
- Messaging systems or sockets connections.
- Front-ending multiple applications with a single user interface.
- Aggregation of data from multiple applications into reports and repositories.

Since there are so many ways to integrate application systems, it can be difficult to select an integration solution that is best for a particular application need. The intent of the Integration Architecture is to define the recommended techniques for integrating heterogeneous application systems and to specify when to implement each technique. The four primary techniques for integration are application integration, electronic data interchange, data access integration, and terminal integration.

Each of the four integration techniques uses a common concept called integration middleware to let new distributed applications coexist and inter-operate with purchased application packages and legacy applications and databases. Integration middleware is the software that provides application connectivity and performs the data transformation and delivery between multiple application systems and databases. (See Figure Below)

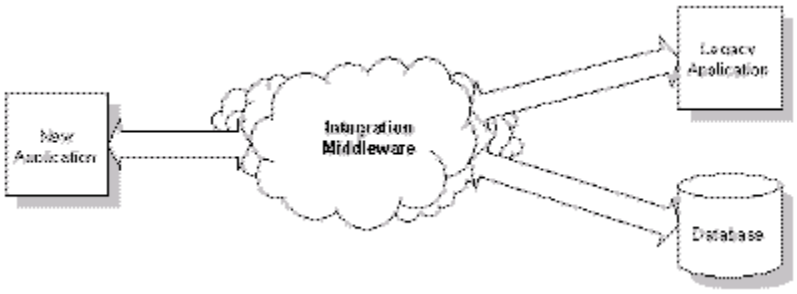

Figure 4. Integration Middleware Solutions

Each integration technique is summarized below, and all but the Data Access will be discussed later in this chapter as a technical topic. Data Access Integration is discussed in Chapter 4: Data Architecture.

Application Integration

Integrates the processing logic or functions from existing systems into new applications. It allows the state to meet the immediate need for allowing online transaction processing (OLTP) legacy systems that were not designed to work together to exchange data and information with other application systems, while keeping the business logic where it was originally developed. This solution prevents the duplication of business logic or functions into the new systems and eliminates the re-keying of data, permitting systems to interact more effectively. An application integration strategy enables the state to retain its investments in the existing legacy systems, and it provides the ability to create a consistent set of platform-independent interfaces that handle common services necessary for multiple systems to work together. These services include data translation, field mapping, and transaction explosion.

Electronic Data Interchange (EDI)

Integrates state applications with related applications operated by vendors or other agencies. Examples of related pairs of applications include purchasing and order entry, billing and accounts payable, student information systems in two different schools, and patient records in hospital admissions and lab applications. EDI is a special case of application integration, since it connects an agency application with an outside application via industry standard data formats.

Data Access Integration

Integrates data from existing systems into new applications. It provides direct communication between databases of applications running on multiple platforms and environments. It improves the usability of existing legacy systems through data interchange between heterogeneous applications. A data access integration strategy allows information to be shared across multiple heterogeneous operational applications without significant impact to the existing technical environment. This integration technique is discussed in significant detail in Chapter 4: Data Architecture.

Terminal Integration

Incorporates a terminal access strategy that allows 3270 terminals to connect to both existing and new client/server applications. 3270 terminals, also known as "dumb terminals," are an integral element of the statewide enterprise. The integration need exists because of this large number of 3270 terminals connected to the legacy SNA network. New applications employ services and user devices that are connected to the TCP/IP network. Terminal integration will allow the 3270 devices to participate with the new applications that are developed within the

TCP/IP communications architecture. Moreover, new client/server applications can be developed and implemented without immediately replacing 3270 terminals.

EDI is a special type of application integration. However, it will be addressed as its own technical topic in this chapter due to the increased interest in EDI as a cross-platform, cross-company, and cross-continent communication solution.

Even though the results are similar from the end user perspective, application integration and data access integration do have inherent differences. Data access integration encompasses data from multiple sources. Application integration encompasses both data and business logic.

Application systems are programmatically integrated when they interact through a program-to-program interface rather than through database-to-database or program-to-database interface. Application integration is usually preferable to data integration for integrating heterogeneous operational application systems. (See Figure Below)

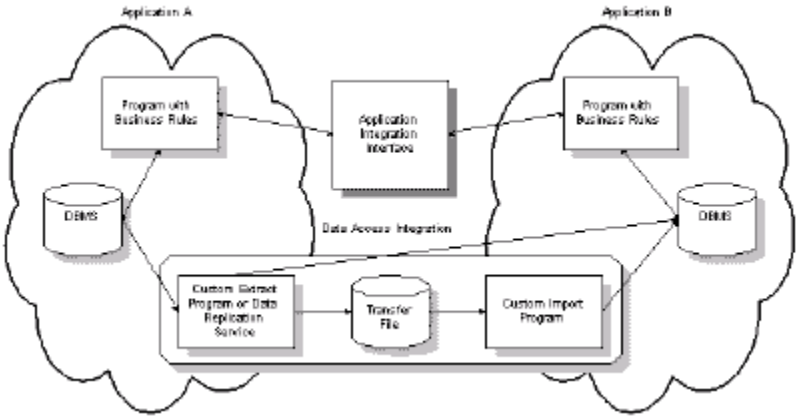

Figure 5. Application Integration vs. Data Access Integration

With application integration techniques applied, the operational databases are accessed only by their own application programs, so the existing business rules are always used. Existing operational applications can be updated or changed without effecting external programs as long as the program-to-program interface remains consistent.

With data access integration, queries can be made to application databases distributed across multiple platforms and database technologies. Database utilities or purchased and custom-developed programs must be used to offload data to other applications and users. Data access integration and application integration can be considered complementary integration tools.